กิติพงษ์ ฤทธิบุตร

Integrated Debugger
นักสืบ Turbo C
(จบ)

โปรแกรมเมอร์ทั้งหลาย ย่อมไม่ปรารถนาให้เกิดบักภายในโปรแกรมแน่นอน มีวิธีกำจัดได้เลย.. ไม่ต้องรอ

ที่ผ่านมาการดีบักของเราจบลงที่การหาบักใน while loop ของฟังก์ชัน 'main ( )' โดยใช้ข้อมูลในบรรทัดแรกของ input file เรียบร้อยแล้ว คราวนี้เราลองหาบักกันต่อใน print loop ซึ่งในลูปนี้จะมีบักอยู่ด้วย แต่การค้นหาบักในคราวนี้คงไม่ง่ายเหมือนกับที่ผ่านมา เพราะเราจำเป็นต้องใช้อินพุตที่ค่อนข้างไม่ปกติ แต่เป็นสิ่งที่อาจจะเกิดขึ้นได้ในการทำงานจริง เช่น ข้อมูลที่เป็นบรรทัดว่าง ซึ่งเป็นตัวอย่างที่ดีที่จะทำให้เรารู้ว่าในการตรวจสอบหาบักของโปรแกรมนั้นควรที่จะใช้ข้อมูลทุกประเภทที่มีโอกาสเกิดขึ้นได้ เพื่อตรวจสอบผลการทำงานของโปรแกรมเมื่อได้รับอินพุตในรูปแบบต่าง ๆ เพราะฉะนั้นเราจะป้อนอินพุตที่เป็นบรรทัดว่างแล้วมาดูกันว่าโปรแกรม WORDCNT จะทำงานและให้ผลลัพธ์ออกมาอย่างไร เป้าหมายใหม่ของเราคือ 'for' loop ที่ทำหน้าที่ในการแสดงเอาต์พุต

หาบักกันต่อ

หลักการของเราได้แก่การเริ่มต้นรันโปรแกรม WORDCNT ตั้งแต่บรรทัดแรกของ 'for' loop ด้วยการเลื่อนเคอร์เซอร์ไปตั้งต้นที่บรรทัดนี้ แต่ไม่กำหนด breakpoint ที่จะให้โปรแกรมหยุดรัน เราจะใช้วิธีการแบบใหม่ในการทำงาน เราสามารถกำหนดตำแหน่งให้โปรแกรมหยุดทำงานได้ด้วยวิธีการที่สะดวกกว่า ในกรณีที่เราต้องการใช้ตำแหน่งหยุดนั้นเพียงครั้งเดียว โดยเลื่อนเคอร์เซอร์ไปที่บรรทัดแรกของ 'for' loop แล้วเลือกใช้คำสั่ง Run/Go to cursor (หรือจะใช้ <F4> ซึ่งเป็นฮอตคีย์ของคำสั่งแทนก็ได้) ผลที่ได้คือดีบักเกอร์จะรันโปรแกรม WORDCNT ไปเรื่อย ๆ จนกว่าจะถึงบรรทัดที่เป็นตำแหน่งของเคอร์เซอร์ ยกเว้นในกรณีที่มีการกำหนด breakpoint ไว้ล่วงหน้า แต่ในตัวอย่างของเราไม่ได้กำหนดไว้ เพราะฉะนั้นการทำงานของคำสั่งนี้จะเป็นไปตามที่กำหนดไว้คือตำแหน่งของเคอร์เซอร์

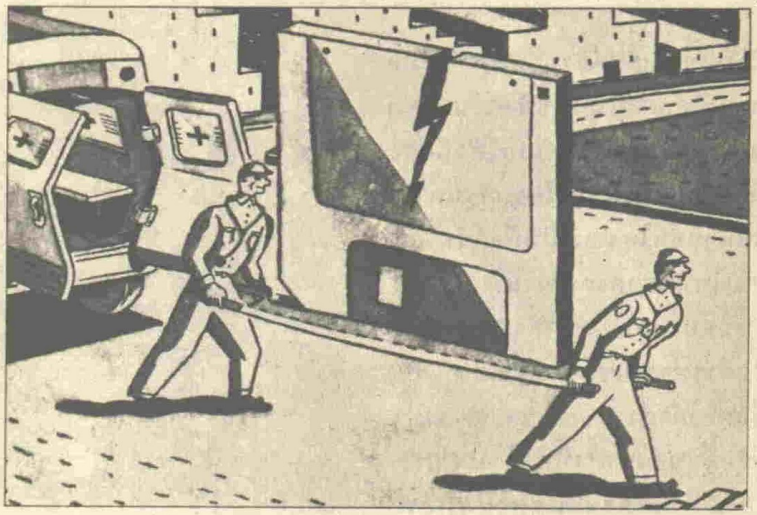
คราวนี้เราสั่งรันต่อไปอีกหนึ่งบรรทัดโดยเลื่อน execution bar ไปที่ฟังก์ชัน 'printf' ในบรรทัดถัดไปของ 'for' loop (จะสังเกตเห็นได้ว่าไม่มีแถบสว่างของ breakpoint บนบรรทัดคำสั่ง 'for' ซึ่งถ้าเราใช้คำสั่งกำหนด breakpoint ไว้ เมื่อเลื่อน execution bar ออกมาแล้วจะปรากฏเครื่องหมายของ breakpoint บนบรรทัดนั้น) และในกรณีนี้คำสั่ง Run/Go to cursor จะเป็นการสั่งรันทีละบรรทัดและไม่มีการกำหนดตำแหน่งของ breakpoint ไว้ใช้ในการรันครั้งหลัง ขั้นต่อไปคือการตรวจสอบการทำงานของ 'for' loop ทีละขั้นด้วยคำสั่ง Run/Trace into คำสั่งนี้จะไม่แสดงขั้นตอนการทำงานของฟังก์ชัน 'printf' ให้เรา

เหมือนกับอีกหลายฟังก์ชัน ทั้งนี้เพราะไม่ได้ define ไว้ในซอร์สไฟล์ที่คอมพายล์ด้วย Debug Information ดังนั้นดีบักเกอร์ได้แต่เพียงรันตามขั้นตอนของฟังก์ชันแล้วแสดงผลลัพธ์ออกมาเท่านั้น นอกจากนี้ยังเป็นเพราะซอร์สไฟล์ของ printf ( ) ไม่ได้อยู่ในดิสค์ของเทอร์โบซีด้วย

ในการรันคำสั่งที่มีการเรียกใช้ printf ( ) แต่ละครั้ง ตัวดีบักเกอร์จะแสดง execution screen ให้ ถ้าต้องการทราบผลลัพธ์ที่ได้เราจำเป็นต้องใช้คำสั่ง Run/User screen (หรือ <Alt><F5> ที่เป็นฮอตคีย์) เพื่อให้ระบบกลับไปใช้ยูสเซอร์สกรีนที่เป็นส่วนของการแสดงผลลัพธ์ของโปรแกรม (ในกรณีนี้คือ WORDCNT) โดยที่บางครั้งอาจพบว่าผลลัพธ์ของโปรแกรมจะถูกเขียนทับด้วยซอร์สโค้ดใน Edit window ซึ่งจะทำให้หายไปได้ด้วยการเรียกใช้คำสั่ง Debug/Refresh display ควรจะตรวจสอบว่าคำสั่ง Debug/Display swapping ถูกกำหนดค่าไว้เป็น Smart หรือ Allways เรียบร้อยแล้ว และต่อจากนี้คงเป็นธุระของเราเองที่จะหาบักและลองแก้ไขให้ถูกต้อง เพราะในโปรแกรม WORDCNT ยังมีบักอีกหลายตัว

Integrated Debugger กับโปรแกรมใหญ่ ๆ

เราได้เรียนรู้ถึงขีดความสามารถของ Integrated Debugger และใช้งานกันบ้างแล้ว แต่ดีบักเกอร์ตัวนี้ยังมีคุณสมบัติอีกหลายอย่างที่จะช่วยให้การดีบักโปรแกรมทำได้สะดวกยิ่งขึ้นไม่ว่าจะเป็นโปรแกรมใหญ่หรือเล็กขนาดไหน ต่อจากนั้นเราจะพิจารณาถึงการใช้ดีบักเกอร์กับการดีบักโปรแกรมขนาดใหญ่ รวมทั้งโปรแกรมที่ต้องใช้หลาย ๆ ไฟล์ร่วมกัน



(multi-file program) ว่าจะทำได้มากน้อยอย่างไร เริ่มต้นด้วยการคอมพายล์โปรแกรม WORDCNT กันใหม่ โดยกำหนดให้ค่าของ Options/Compiler/Code generation/Standard stack frame เป็น On ถ้าโปรแกรมถูกโหลดเข้ามาไว้ใน Edit window เรียบร้อยแล้วก็สั่งคอมพายล์ได้เลย (ใช้ <Alt><F9>) แล้วเริ่มงานดีบักของเราด้วยการเรียกใช้คำสั่ง Run/Step over หรือใช้ฮอตคีย์ <F8> ทีนี้ทุกอย่างก็พร้อมที่จะทำงานในขั้นต่อไป

ว่าด้วยเรื่องของฟังก์ชัน

สิ่งที่เราจะกล่าวถึงก่อนคือคำสั่ง Debug/Find Function ซึ่งเป็นคำสั่งให้ดีบักเกอร์แสดงรายละเอียดของฟังก์ชัน (function's definition) ใน Edit window สำหรับขีดความสามารถของคำสั่งนี้เป็นการค้นหาฟังก์ชันต่าง ๆ ที่อยู่ในโปรแกรมและถูกคอมพายล์ด้วยคำสั่ง Debug/Source Debugging และ O/C/C-/OBJ Debugging information ที่กำหนดให้เป็น On จึงนับได้ว่าเป็นคำสั่งที่มีประโยชน์มากคำสั่งหนึ่งของดีบักเกอร์ตัวนี้ โดยเฉพาะอย่างยิ่งในกรณีที่พบบักขณะกำลังทำงานอยู่ในส่วนหนึ่งของโปรแกรม แต่มีความจำเป็นต้องแก้ไขบักโดยการแก้ไขโปรแกรมส่วนอื่นที่ทำงานเกี่ยวข้องกัน หรือในกรณีที่ต้องการให้แสดงคำสั่งของฟังก์ชัน คำแนะนำเพื่อช่วยเตือนความจำเกี่ยวกับวิธีการทำงานของฟังก์ชันนั้น ๆ เราลองใช้คำสั่งนี้ด้วยการเลื่อนเคอร์เซอร์ไปยังบรรทัดคำสั่งที่มีการเรียกใช้ฟังก์ชัน 'wordlen' ใน main ( ) เรียกใช้คำสั่ง Run/Go to cursor แล้วเลื่อนเคอร์เซอร์ไปที่คำว่า wordlen และใช้คำสั่ง Debug/Find Function ส่งผลให้ดีบักเกอร์เปิดวินโดว์เพื่อให้เราใส่ชื่อฟังก์ชันที่ต้องการ เราก็พิมพ์ 'wordlen' แล้วกด <Enter> สั่งให้ดีบักเกอร์แสดงรายละเอียดคำสั่งของ wordlen ( ) ใน Edit window

ใช้บริการจาก stack

การเขียนโปรแกรมขนาดใหญ่ด้วยภาษาซี สิ่งที่จะต้องเกิดขึ้นอย่างแน่นอนคือการเรียกใช้ฟังก์ชันต่าง ๆ เป็นจำนวนมากจากฟังก์ชันหนึ่งไปอีกฟังก์ชันหนึ่ง หมายความว่าการทำงานของฟังก์ชันหนึ่งอาจมีการเรียกใช้ฟังก์ชันอื่นอีกหลายชุดตามลำดับขั้นตอนการทำงานของโปรแกรม ทำให้เกิดความสับสนเรื่องลำดับการทำงานของฟังก์ชันเหล่านั้น

ได้ บริษัทบอร์แลนด์จึงบรรจุคุณสมบัติในการจัดลำดับการเรียกใช้ฟังก์ชันต่าง ๆ ของโปรแกรมให้กับดีบักเกอร์ตัวนี้ด้วย ผู้ใช้จะทราบว่าขณะนั้นโปรแกรมกำลังเรียกใช้ฟังก์ชันใด และเป็นฟังก์ชันที่อยู่ในลำดับไหนของการทำงาน

ขั้นต่อไปลองใช้คำสั่ง Trace into เข้าไปดูการทำงานของ wordlen ( ) แล้วเรียกใช้คำสั่ง Debug/Call stack หรือฮอตคีย์ของคำสั่งนี้คือ <Ctrl> <F3> ดีบักเกอร์จะแสดงป๊อปอัพ-วินโดว์ที่ประกอบด้วยรายชื่อของฟังก์ชันในโปรแกรม WORDCNT. ที่ถูกเรียกใช้โดยฟังก์ชันที่กำลังใช้งานนั้นจะอยู่ส่วนบนสุดของสแต็ก ส่วน main ( ) จะอยู่ด้านล่างของสแต็ก นอกจากนี้ในวินโดว์ของ Call stack จะแสดงค่าพารามิเตอร์ต่าง ๆ ที่ฟังก์ชันแต่ละชุดเรียกใช้ด้วย นอกเหนือจากชื่อของฟังก์ชันที่แสดงไว้ในการใช้งานสแต็ก เราสามารถบอกให้ระบบแสดงบรรทัดคำสั่งของฟังก์ชันที่ประมวลผลในขณะนั้น โดยการเลื่อนแถบสว่างของ Call stack window ไปยังฟังก์ชันที่ต้องการ และกด <Enter> เช่น ถ้าเราต้องการให้แสดงบรรทัดคำสั่งของ main ( ) ก็เลื่อนแถบสว่างไปที่ main ( ) และกดแป้น <Enter> ซึ่งจะทำให้ดีบักเกอร์ดึงส่วนของโปรแกรมที่มีบรรทัดที่เป็นชุดคำสั่งของ main ( ) ที่เรียกใช้ Wordlen ( ) เข้ามาใน Edit window หากยังมีการเรียกใช้ฟังก์ชันอื่นอีกก็สามารถสั่งให้ระบบแสดงบรรทัดที่กำลังประมวลผลของฟังก์ชันนั้นได้ด้วยการเรียกใช้คำสั่ง Debug/Call stack อีกครั้ง แล้วเลือกชื่อฟังก์ชันที่ต้องการใหม่ เมื่อต้องการจะกลับไปที่ execution bar หลังจากที่ได้เรียกดูส่วนต่าง ๆ ของโปรแกรมเสร็จแล้ว เราทำได้ด้วยการใช้คำสั่ง Debug/Call stack เช่นกัน โดยเลื่อนแถบสว่างของ Call stack window ไปยังฟังก์ชันบนสุดของสแต็ก แล้วกด <Enter> ระบบจะพากลับไปสู่ execution bar ใน Edit window

เมื่อใช้งานกับ multiple source files

นอกจาก Integrated Debugger จะใช้งานกับซอร์สไฟล์ทั้งขนาดเล็กและใหญ่ได้แล้ว ยังสามารถใช้งานกับโปรแกรมที่มีซอร์สไฟล์หลายไฟล์ได้อย่างดี ไม่ว่าจะเป็นการเรียกใช้คำสั่งไหนของดีบักเกอร์ เช่น ถ้าใช้คำสั่ง Debug/Find Function เพื่อค้นหาฟังก์ชันที่ define เอาไว้ในซอร์สไฟล์อื่นที่ไม่ใช่ไฟล์ภายใน Edit window ตัวดีบักเกอร์จะทำการโหลดไฟล์ที่มีฟังก์ชันที่ต้องการเก็บไว้ใน Edit window แทน และถ้าไฟล์ที่กำลังใช้งานอยู่นั้นถูกแก้ไขแล้ว ดีบักเกอร์จะถามว่าต้องการเก็บไฟล์ที่แก้ไขนั้นไว้หรือไม่ก่อนโหลดไฟล์ใหม่เข้ามา

ในทำนองเดียวกันถ้าเรียกใช้คำสั่ง Call stack (ในคำสั่ง Debug/Call stack) และเลือกดูฟังก์ชันที่ไม่ได้อยู่ในซอร์สไฟล์ใน Edit window ตัวดีบักเกอร์จะโหลดซอร์สไฟล์ชุดใหม่ของฟังก์ชันที่เลือกดูเข้ามาแทน โดย execution bar จะอยู่ที่บรรทัดซึ่งเป็นคำสั่งต่อไปที่จะทำการประมวลผลของฟังก์ชันนั้น และถ้าผู้ใช้แก้ไขซอร์สไฟล์ต่าง ๆ ที่เรียกขึ้นมา ดีบักเกอร์จะถามทุกครั้งว่าต้องการเก็บสิ่งที่แก้ไขไว้หรือไม่ก่อนทำการโหลดไฟล์ชุดเก่าขึ้นมาใช้ อย่างไรก็ตาม ถ้าโปรแกรมที่กำลังดีบักนั้นประกอบด้วยซอร์สไฟล์จำนวนมาก เราควรจะดีบักเพียงครั้งละ 2-3 ไฟล์ เพื่อความสะดวกในการควบคุมการทำงานของโปรแกรมเมื่อมีการแก้ไขบางส่วนของโปรแกรม

ดีบักกับแอสเซมบลีโค้ด

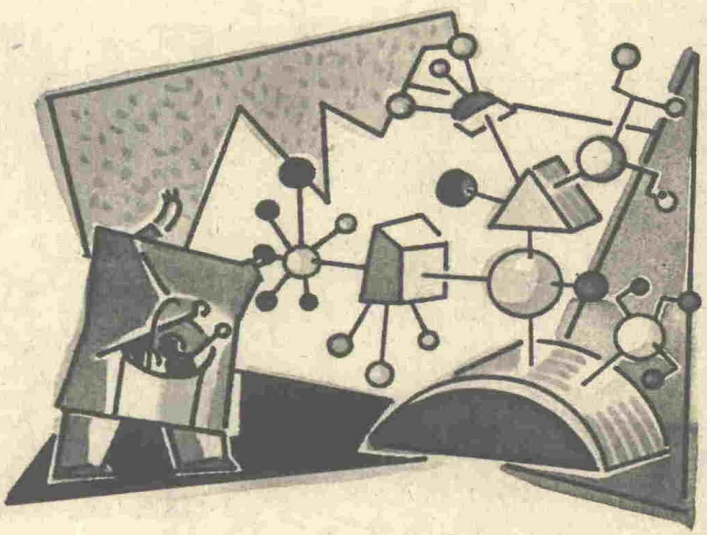
ถึงแม้ว่าบริษัทบอร์แลนด์จะสร้าง Integrated Debugger ให้เป็นดีบักเกอร์ที่มีวัตถุประสงค์เพื่อการดีบักโปรแกรมที่มีซอร์สโค้ดของเทอร์โบซี แต่ได้รวมขีดความสามารถในการดีบักโปรแกรมที่เรียกใช้ภาษาแอสเซมบลีได้ด้วย เราสามารถใช้เทอร์โบซีใน integrated development environment ร่วมกับ TASM assembler ของบอร์แลนด์ ซึ่งจะทำให้สามารถติดตามตรวจสอบโมดูลภาษาแอสเซมบลีที่นำมาใช้กับเทอร์โบซีด้วยการใช้คำสั่ง -zi ของ TASM ทำให้เทอร์โบซีสามารถทำงานกับคำสั่งต่าง ๆ ของแอสเซมบลีได้ และสามารถแสดง

ชุดคำสั่งต่าง ๆ ในภาษาแอสเซมบลีของฟังก์ชันนั้น ส่วนการดีบักสามารถใช้คำสั่งต่าง ๆ ในดีบักเมนูได้ทั้งหมด เช่น Debug/Go to cursor (<F4>), Debug/Trace into (<F7>) และ Debug/Step over (<F8>) เป็นต้น

นอกจากนี้สัญลักษณ์ต่าง ๆ ที่ใช้ภาษาแอสเซมบลีนั้นสามารถนำมาใช้ในการทดสอบการประมวลผล expression และ Watch expression ได้ด้วย รวมถึงการนำ pseudo-register (เช่น __AX และ __BX เป็นต้น) และตัวแปรพิเศษ เช่น __FLAGS ที่มีผลต่อการทำงานของ Flag register ของซีพียูมาใช้ได้เช่นกัน อย่างไรก็ตาม เนื่องจากตัว Integrated Debugger ถูกออกแบบมาเพื่อใช้กับภาษาซี เพราะฉะนั้นขีดความสามารถในการทำงานกับแอสเซมบลีซอร์สโค้ดจึงมีขอบเขตจำกัดบ้างเล็กน้อย ไม่สามารถใช้กับทุกคำสั่งของแอสเซมบลี ดังนั้นในกรณีที่ต้องการดีบักกับซอร์สโค้ดที่เป็นแอสเซมบลีจริง ๆ ควรจะใช้ดีบักเกอร์ตัวอื่น เช่น Standalone Debugger ของบริษัทบอร์แลนด์ เป็นต้น

แนวทางทดสอบซอฟต์แวร์

การทดสอบการทำงานโปรแกรมที่เขียนขึ้นนั้น นอกเหนือจากการที่จะต้องฝึกฝนให้สามารถใช้งานดีบักเกอร์ได้อย่างมีประสิทธิภาพแล้ว ควรฝึกหรือเสริมความชำนาญเรื่องความสามารถในการวิเคราะห์หรือการชี้จุดที่ทำให้โปรแกรมทำงานผิดพลาดด้วย เพราะความสามารถในลักษณะนี้จะช่วยให้การดีบักโปรแกรมทำได้รวดเร็วขึ้น โดยที่เราสามารถจะลงมือค้นหาและแก้ไขบักได้ในตำแหน่งที่เกิดขึ้นได้เลย แทนที่จะต้อง 'งม' หานานเกินไป ทำให้เสียเวลาอย่างมาก (ชวนปวดหัวอีกต่างหาก) และความสามารถที่ว่านี้เป็นเรื่องที่ฝึกฝนกันได้ ไม่ว่าจะเป็นการศึกษาค้นคว้าด้วยตนเอง เรียนรู้จากประสบการณ์ของคนอื่น หรือใช้ร่วมกันทั้งสองอย่าง อย่าลืมคำพังเพยของคนโบราณที่ว่า "อายครูบ่รู้วิชา" ก็แล้วกัน ผู้เขียนเชื่อว่ามีคนอีกมากมายที่ไม่รู้เหมือนเรานั่นแหละ เพราะฉะนั้นเราจะมาคุยกันถึงวิธีการหรือแนวทางที่จะช่วยให้การทดสอบการทำงานของโปรแกรมทำได้ง่ายขึ้น



กำหนดวิธีการทำงาน

ความจริงในเรื่องนี้คงจะไม่มีมาตรฐานให้ทุกคนนำไปใช้ได้ เพราะการกำหนดวิธีการหรือขั้นตอนการทำงานย่อมเป็นไปตามความพอใจและประสบการณ์ของแต่ละบุคคล รวมถึงลักษณะของโปรแกรมที่ทำการทดสอบ (คงจะเหมือนกันได้ยาก) อย่างไรก็ตาม ในการตรวจสอบหาบักของโปรแกรมนั้น ขั้นตอนที่เป็นพื้นฐานที่ไม่ควรจะละเลยอาจประกอบด้วยวิธีการต่อไปนี้

- ป้อนข้อมูล (input) ที่ใช้ตามปกติของโปรแกรม แล้วติดตามดูการทำงานของโปรแกรมด้วยคำสั่ง Debug/Evaluate และผลลัพธ์ที่ออกมาว่าถูกต้องหรือไม่ ถ้าพบบักก็ทำการแก้ไขให้ถูกต้อง
- ป้อนข้อมูลเพื่อทดสอบการทำงานของโปรแกรมในส่วนที่ไม่สามารถตรวจสอบการทำงานได้
- ทดสอบการทำงานของทุกคำสั่งในโปรแกรม ซึ่งจะช่วยให้เราสามารถตรวจสอบบักในส่วนใดส่วนหนึ่งของโปรแกรมที่เราไม่เคยคิดว่าจะเกิดปัญหาขึ้น เช่น ในโปรแกรมตัวอย่าง WORDCNT การใช้วิธีนี้ทำให้ตรวจพบชุดคำสั่งที่เขียนผิดไว้ คือมีเครื่องหมาย ';' (semicolon) อยู่หลังคำสั่ง 'else' (การใช้วิธีอื่นอาจพบข้อผิดพลาดได้ยาก เพราะในกรณีนี้เครื่องหมาย ';' อยู่ที่ท้ายบรรทัดของคำสั่งใน Edit window ห่างจากชุดคำสั่งทำให้ถูกละเลยได้ง่าย เป็นผลให้โปรแกรมทำงานผิดพลาดเมื่อตรวจพบคำที่มีค่ามากกว่า MAXWORDLEN
- จะต้องให้ความสนใจเป็นพิเศษกับคำสั่งหรือ expression ที่มีทางออกหลายทาง เช่น

```
if (strcmp (a,b))...
```

เพราะ strcmp สามารถให้ค่าได้ 3 ค่าคือ

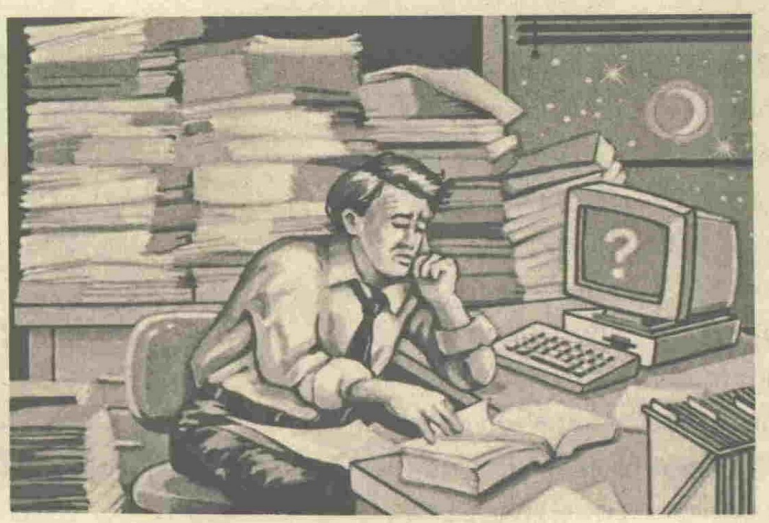
1. 0 เมื่อ a มีค่าเท่ากับ b
2. −1 เมื่อ a มีค่าน้อยกว่า b
3. +1 เมื่อ a มีค่ามากกว่า b

ฉะนั้นในการทดสอบควรจะป้อนข้อมูลให้ได้ผลลัพธ์ทั้งสามแบบ เพื่อให้แน่ใจว่าโปรแกรมสามารถทำงานได้อย่างถูกต้องในทุก ๆ เงื่อนไข นอกจากนี้ก็ยังมีคำสั่งอื่นอีก เช่น

```
x = (x>0) ? func (x) : 0;
```

คำสั่งนี้ประกอบด้วย 'implicit if' ที่จะให้ผลลัพธ์แตกต่างกันได้ 2 ค่า เป็นต้น

- ให้ความสนใจเป็นพิเศษกับการทำงานของโปรแกรมที่มีเงื่อนไขซึ่งกำหนดขอบเขตการทำงาน เช่น เงื่อนไขที่จะทำให้โปรแกรมหลุดออกจากการทำงานในลูป หรือใส่ข้อมูลลงในอะเรย์และอื่น ๆ ในลักษณะนี้ ความผิดพลาดเหล่านี้มักจะเกิดขึ้นได้เสมอ จากตัวอย่างของเราจะมีให้เห็นสองแห่งที่ทำให้เกิดบักในโปรแกรมตามกรณีนี้ จุดแรกก็คือ 'for' loop ชุดสุดท้ายของโปรแกรมที่ไม่ยอมพิมพ์ค่าของ 'wordlencnt' ที่ทำหน้าที่นับจำนวนตัวอักษรของคำที่ยาวที่สุดใน MAXWORDLEN จุดที่สองคือการกำหนดค่าให้กับ WORDLENCNT น้อยกว่าความเป็นจริง ทำให้เกิดความผิดพลาดเมื่อมีข้อมูลยาวกว่าที่กำหนดไว้



- ทดสอบการทำงานของโปรแกรมทั้งชุดว่าทำงานได้ถูกต้องหรือไม่ ถ้าโปรแกรมที่ทดสอบเป็นโปรแกรมที่ต้องการให้คนอื่นนำไปใช้ เราควรจะทดสอบทุก ๆ รูปแบบของความผิดพลาดที่อาจเกิดขึ้นได้ โปรแกรมที่ผ่านการทดสอบความผิดพลาดในลักษณะต่าง ๆ มากเพียงใดย่อมมีโอกาสทำงานผิดพลาดน้อย
- ในกรณีที่เขียนโปรแกรมให้คนอื่นใช้ควรจะหาผู้ช่วยทดสอบการทำงาน และควรจะเป็นคนที่มีคุณสมบัติเป็นผู้ใช้มากกว่าโปรแกรมเมอร์ด้วยกันเอง

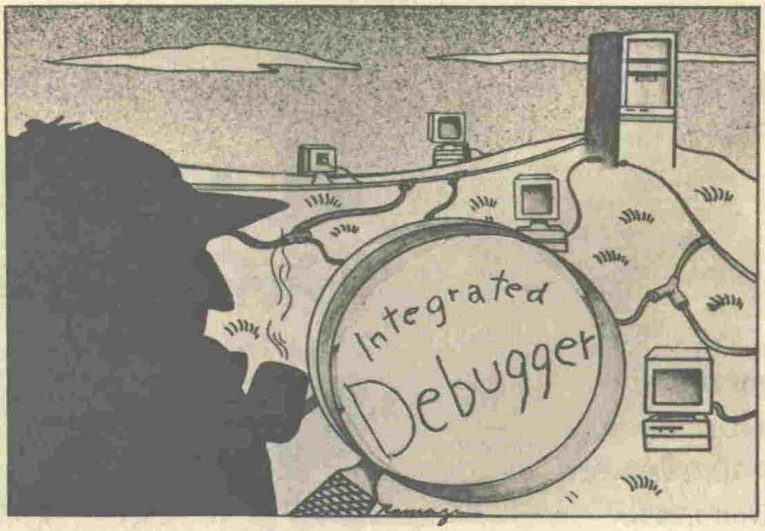
ทดสอบหลังการแก้ไข

ในการดีบักโปรแกรม หากมีส่วนใดของโปรแกรมถูกแก้ไข เราจำเป็นจะต้องทดสอบการทำงานของโปรแกรมในส่วนนั้น รวมถึงโปรแกรมในส่วนที่ทำงานเกี่ยวข้องกับส่วนของโปรแกรมที่ถูกแก้ไขด้วย และถ้าโปรแกรมที่เขียนขึ้นมีความซับซ้อนมากควรจะบันทึกผลการทดสอบที่ได้ทำไปแล้ว เพราะจะเป็นประโยชน์ต่อการแก้ไขโปรแกรมในส่วนอื่น ๆ ต่อไป ถ้าการทดสอบมีการใช้ข้อมูล (input files) ใดเป็นการเฉพาะควรจะเก็บไฟล์เหล่านั้นไว้ด้วย เพื่อจะได้นำมาใช้ในการทดสอบการทำงานของโปรแกรมในโอกาสต่อไป

การออกแบบโปรแกรม

วิธีการหลีกเลี่ยงบักอย่างหนึ่งเห็นจะได้แก่การออกแบบโปรแกรมด้วยความละเอียดรอบคอบ ซึ่งวิธีการก็คือการเขียนคำสั่งให้ชัดเจน เช่น มีย่อหน้าใช้ comment และคำอธิบายตัวแปร รวมถึงการเขียนคำสั่งด้วยวิธีธรรมดา พยายามหลีกเลี่ยงการเขียนชุดคำสั่งที่ซับซ้อนที่สามารถเขียนได้ด้วยคำสั่งธรรมดาหลาย ๆ ชุด เพราะเทอร์โบซีจะทำการ optimize code ให้เอง และซอร์สโปรแกรมที่ได้จะมีลักษณะที่อ่านง่าย สะดวกต่อการดีบักและแก้ไขหรือพัฒนาโปรแกรม นอกจากนี้เราควรเขียนโปรแกรมที่มักเรียกใช้ฟังก์ชันที่ทำงานไม่ซับซ้อนเกินไป ใช้โครงสร้างการทำงานที่ดี เพราะจะช่วยให้สามารถทดสอบการทำงานและผลลัพธ์ของโปรแกรมได้ไม่ยาก ช่วยให้โปรแกรมอ่านง่าย แก้ไขง่าย

สำหรับการใช้งานฟังก์ชันควร



จะหาทางลดจำนวนของ data element ที่ฟังก์ชันต้องการใช้ รวมถึงจำนวนของ element ที่จะเปลี่ยนแปลงไปเมื่อมีการเรียกใช้ฟังก์ชันด้วย นอกจากจะช่วยให้สามารถทดสอบและแก้ไขได้ง่ายแล้ว ยังช่วยลดโอกาสที่ฟังก์ชันจะทำงานผิดพลาดได้อีกส่วนหนึ่ง ประการสำคัญคือไม่ควรเปลี่ยนแปลงส่วนของโปรแกรมที่ทำงานได้อย่างมีประสิทธิภาพแล้ว ถึงแม้ว่าอาจจะทำให้โปรแกรมสามารถทำงานได้ดีขึ้นกว่าเดิมบ้าง แต่จะมีผลให้ซอร์สโปรแกรมที่ได้อ่านยากขึ้นและลำบากในการดีบัก สำหรับในกรณีที่ปรากฏว่าโปรแกรมทำงานช้าเกินไป ควรจะพิจารณาเป็นการเฉพาะว่าส่วนไหนของโปรแกรมที่ทำให้โปรแกรมทำงานช้า เพื่อทำการแก้ไขเฉพาะในส่วนที่เกี่ยวข้องจริงเท่านั้น สุดท้ายก็คือพยายามเขียนฟังก์ชันที่สามารถใช้งานทั่ว ๆ ไปได้ เพื่อให้ส่วนอื่นของโปรแกรมสามารถใช้ประโยชน์จากฟังก์ชันที่เขียนขึ้นได้มากที่สุด ซึ่งวิธีการเขียนฟังก์ชันเพื่อใช้งานทั่ว ๆ ไปนั้นมักจะเขียนและดีบักได้ง่ายกว่าการเขียนฟังก์ชันการทำงานเฉพาะอย่าง

รูปแบบการดีบัก

วิธีการดีบักโปรแกรมนั้นก็เหมือนกับวิธีการเขียนโปรแกรมคือสามารถทำได้ทั้งแบบจากบนลงล่าง (Top down) และจากล่างขึ้นบน (Bottom up) แต่วิธีการที่น่าจะดีกว่าก็คือการดีบักด้วยวิธีหลัง ซึ่งจะเป็นการทดสอบการทำงานของโปรแกรมจากฟังก์ชันในระดับล่างสุดคือฟังก์ชันที่ไม่ได้เรียกใช้ฟังก์ชันอื่นอีก แล้วจึงทำการทดสอบฟังก์ชันในระดับที่อยู่เหนือขึ้นไปเรื่อย ๆ จนถึง main ( ) เพราะวิธีการแบบนี้จะช่วยให้การตรวจสอบการทำงานของฟังก์ชันต่าง ๆ ในระดับเหนือกว่าเป็นไปด้วยความมั่นใจยิ่งขึ้นว่า ฟังก์ชันที่เรียกใช้นั้นทำงานได้ถูกต้องแล้ว ถ้ามีการทำงานผิดพลาดแสดงว่าเกิดจากการทำงานในส่วนอื่นของฟังก์ชัน ไม่ได้เกิดจากการเรียกใช้ฟังก์ชันในระดับต่ำ

หาบักประเภทเดียวกัน

เมื่อมีการตรวจพบบักในโปรแกรม สิ่งที่ควรจะทำอย่างหนึ่งคือ ลองตรวจหาบักจากส่วนอื่นของโปรแกรมที่มีลักษณะการทำงานเหมือนกัน เพราะเป็นไปได้ว่าบักที่เกิดขึ้นนั้นอาจจะเกิดขึ้นอีกในส่วนอื่นของโปรแกรม เช่น ถ้าพบว่ามีการเรียกใช้ฟังก์ชันตามตัวอย่าง

```
fp = fopen ("rb", filename);
```

เป็นการเขียนคำสั่งผิดไปจากที่ควรคือ

```
fp = fopen (filename, "rb");
```

ย่อมเป็นไปได้ที่จะมีการเขียนคำสั่งนี้ผิดในส่วนอื่น ๆ ของโปรแกรมด้วย เพราะฉะนั้นเราควรตรวจดูทุกจุดของโปรแกรมที่มีการเรียกใช้ฟังก์ชัน 'fopen' หรือฟังก์ชันอื่นที่มีลักษณะการเขียนคำสั่งเหมือนกันหรือคล้ายคลึงกัน

ส่งท้าย

เป็นอันว่าเรื่องราวของ Integrated Debugger ของบริษัทบอร์แลนด์ได้จบลงด้วยประการฉะนี้ แต่สิ่งต่าง ๆ ที่เราได้พูดถึงนั้นเป็นเพียงแนวทางในการใช้งานดีบักเกอร์ตัวนี้เท่านั้น ส่วนความสำเร็จหรือความชำนาญในการนำไปใช้งานอย่างจริงจังคงจะต้องอาศัยความมานะพยายามของแต่ละคน ผู้เขียนคิดว่าน่าจะเกินคุ้มถ้าหากผู้อ่านชอบเขียนโปรแกรมภาษาซีด้วยเทอร์โบซี หรือแม้แต่จะเป็นการฝึกหัดใช้งานดีบักเกอร์ตัวอื่น เพราะความเชื่อมั่นในวิธีการหรือขั้นตอนการเขียนโปรแกรมที่ดีนั้นคงไม่เพียงพอโดยเฉพาะอย่างยิ่งกับโปรแกรมที่ต้องใช้กับงานสำคัญมาก ๆ นอกจากนี้คงจะไม่มีโปรแกรมเมอร์คนใดที่เขียนโปรแกรมขึ้นมาแล้วอยากให้คนอื่น (ที่เอาโปรแกรมไปใช้) สบประมาทตนได้ว่ามีแต่บัก ใช้งานไม่ได้หรืออะไรทำนองนี้แน่นอน เพราะนอกจากจะสะท้อนให้เห็นถึงฝีมือในการเขียนโปรแกรมแล้ว ยังเป็นเรื่องของความรับผิดชอบต่อผลงานที่ทำด้วย ฉะนั้นเครื่องมือนี้คงจะช่วยรับภาระและกำจัดความไม่น่าอภิรมย์ในส่วนนี้ออกไปได้